

# Trellis-Based Check Node Processing for Low-Complexity Nonbinary LP Decoding

Mayur Punekar and Mark F. Flanagan

Claude Shannon Institute,

University College Dublin, Belfield, Dublin 4, Ireland.

{mayur.punekar, mark.flanagan}@ieee.org

**Abstract**—Linear Programming (LP) decoding is emerging as an attractive alternative to decode Low-Density Parity-Check (LDPC) codes. However, the earliest LP decoders proposed for binary and nonbinary LDPC codes are not suitable for use at moderate and large code lengths. To overcome this problem, Vontobel *et al.* developed an iterative Low-Complexity LP (LCLP) decoding algorithm for binary LDPC codes. The variable and check node calculations of binary LCLP decoding algorithm are related to those of binary Belief Propagation (BP). The present authors generalized this work to derive an iterative LCLP decoding algorithm for nonbinary linear codes. Contrary to binary LCLP, the variable and check node calculations of this algorithm are in general different from that of nonbinary BP. The overall complexity of nonbinary LCLP decoding is linear in block length; however the complexity of its check node calculations is exponential in the check node degree. In this paper, we propose a modified BCJR algorithm for efficient check node processing in the nonbinary LCLP decoding algorithm. The proposed algorithm has complexity linear in the check node degree. We also introduce an alternative state metric to improve the run time of the proposed algorithm. Simulation results are presented for (504, 252) and (1008, 504) nonbinary LDPC codes over  $\mathbb{Z}_4$ .

## I. INTRODUCTION

Binary and nonbinary LDPC codes [1] have attracted much attention in the research community in the past decade. LDPC codes are generally decoded by the iterative BP algorithm which performs remarkably well at moderate SNR levels. Due to their capacity achieving performance, LDPC codes are used in many current communications systems. They are also a promising candidate for future high data rate communication systems as well as for memory applications. However, BP suffers from a so called *error floor* problem at high SNR. Also, the heuristic nature of BP makes it difficult to analyze, and simulations are too time consuming for the prediction of the error floor.

In recent years, the new approach of LP decoding is emerging as an attractive alternative to the BP decoding. LP decoding for binary LDPC codes was proposed by Feldman *et al.* [2]. In LP decoding, the maximum likelihood decoding problem is modeled as an LP problem. In contrast to BP decoding, LP decoding relies on a well studied branch of mathematics which provides a basis for better understanding of the decoding algorithms. The work of [4] extended the LP decoding framework of Feldman *et al.* to nonbinary linear codes. Binary and nonbinary LP decoding algorithms rely on standard LP solvers based on simplex or interior point

methods. However, the time complexity of these solvers is known to be exponential in number of variables, which limits the use of LP decoding to codes of small block length. To decode longer codes, a specialized low complexity LP decoding algorithm is necessary. Such a low-complexity algorithm for binary LDPC codes was proposed by Vontobel *et al.* in [3]. The present authors, in [5], extended the binary LCLP decoding algorithm [3] to nonbinary codes. The complexity of the proposed nonbinary LCLP decoding algorithm is linear in the block length. As opposed to binary LCLP decoding, nonbinary LCLP decoding is not directly related to nonbinary BP. Due to this, the complexity of the check node calculations of nonbinary LCLP decoding is exponential in the maximum check node degree. In this paper, we propose a modified BCJR algorithm for the check node processing of nonbinary LCLP decoding. The proposed algorithm has complexity linear in the check node degree and allows for efficient implementation of nonbinary LCLP decoding. We also propose an alternative state metric which can be used for faster check node processing.

This paper is organized as follows. Notation and background information is given in Section II. Section III reviews the nonbinary LCLP decoding algorithm from [5]. Section IV contains the modified BCJR algorithm for check node processing and also explains the alternative state metric. Section V presents the simulation results, and Section VI concludes the paper.

## II. NOTATION AND BACKGROUND

Let  $\mathbb{R}$  be a finite ring with  $q$  elements with 0 as its additive identity. We define  $\mathbb{R}^- = \mathbb{R} \setminus \{0\}$ . Let  $\mathcal{C}$  be a linear code of length  $n$  over the ring  $\mathbb{R}$ , defined by  $\mathcal{C} = \{c \in \mathbb{R}^n : c\mathcal{H}^T = \mathbf{0}\}$ , where  $\mathcal{H}$  is a  $m \times n$  parity-check matrix with entries from  $\mathbb{R}$ .  $R(\mathcal{C}) = \log_q(|\mathcal{C}|)/n$  is the rate of code  $\mathcal{C}$ . Hence, the code  $\mathcal{C}$  is an  $[n, \log_q(|\mathcal{C}|)]$  linear code over  $\mathbb{R}$ . The row indices and column indices of  $\mathcal{H}$  are denoted by the sets  $\mathcal{J} = \{1, \dots, m\}$  and  $\mathcal{I} = \{1, \dots, n\}$  respectively. The  $j$ -th row of  $\mathcal{H}$  is denoted by  $\mathcal{H}_j$  and the  $i$ -th column of  $\mathcal{H}$  is denoted by  $\mathcal{H}^i$ .  $\text{supp}(c)$  denotes the support of the vector  $c$ . For each  $j \in \mathcal{J}$ , let  $\mathcal{I}_j = \text{supp}(\mathcal{H}_j)$  and for each  $i \in \mathcal{I}$ , let  $\mathcal{J}_i = \text{supp}(\mathcal{H}^i)$ . Also let  $d_j = |\mathcal{I}_j|$  and  $d = \max_{j \in \mathcal{J}} \{d_j\}$ . We define the set  $\mathcal{E} = \{(i, j) \in \mathcal{I} \times \mathcal{J} : j \in \mathcal{J}, i \in \mathcal{I}_j\} = \{(i, j) \in \mathcal{I} \times \mathcal{J} : i \in \mathcal{I}, j \in \mathcal{J}_i\}$ . Moreover for each  $j \in \mathcal{J}$

we define the local Single Parity Check (SPC) code

$$\mathcal{C}_j = \left\{ (b_i)_{i \in \mathcal{I}_j} : \sum_{i \in \mathcal{I}_j} b_i \cdot \mathcal{H}_{j,i} = 0 \right\}$$

For each  $i \in \mathcal{I}$ ,  $\mathcal{A}_i \subseteq \mathbb{R}^{|\{0\} \cup \mathcal{J}_i|}$  denotes the repetition code of the appropriate length and indexing. We also use variables  $\mathbf{u}_{i,j} = (u_{i,j}^{(\alpha)})_{\alpha \in \mathbb{R}^-}$  and  $\mathbf{v}_{j,i} = (v_{j,i}^{(\alpha)})_{\alpha \in \mathbb{R}^-}$  for all  $i \in \mathcal{I}$ ,  $j \in \mathcal{J}_i \cup \{0\}$ ; also for  $i \in \mathcal{I}$ ,  $\mathbf{u}_i = (\mathbf{u}_{i,j})_{j \in \mathcal{J}_i \cup \{0\}}$  and similarly for  $j \in \mathcal{J}$ ,  $\mathbf{v}_j = (\mathbf{v}_{j,i})_{i \in \mathcal{I}_j}$ .

We use the following mapping given in [4],

$$\xi : \mathbb{R} \rightarrow \{0, 1\}^{q-1} \subset \mathbb{R}^{q-1}$$

by

$$\xi(\alpha) = \mathbf{x} = (x^{(\rho)})_{\rho \in \mathbb{R}^-}$$

such that, for each  $\rho \in \mathbb{R}^-$

$$x^{(\rho)} = \begin{cases} 1, & \text{if } \rho = \alpha \\ 0, & \text{otherwise} \end{cases}$$

We extend this mapping to define

$$\Xi : \bigcup_{t \in \mathbb{Z}^+} \mathbb{R}^t \rightarrow \bigcup_{t \in \mathbb{Z}^+} \{0, 1\}^{(q-1)t} \subset \bigcup_{t \in \mathbb{Z}^+} \mathbb{R}^{(q-1)t},$$

where,

$$\Xi(\mathbf{c}) = (\xi(c_1), \dots, \xi(c_t)), \quad \forall \mathbf{c} \in \mathbb{R}^t, t \in \mathbb{Z}^+.$$

For  $\kappa \in \mathbb{R}$ ,  $\kappa > 0$ , we define the function  $\psi(x) = e^{\kappa x}$  and its inverse  $\psi^{-1}(x) = \frac{1}{\kappa} \log(x)$ . We also use the soft-minimum operator introduced in [3]. For any  $\kappa \in \mathbb{R}$ ,  $\kappa > 0$ , the soft-minimum operator is defined as

$$\min_l^{(\kappa)} \{z_l\} \triangleq -\frac{1}{\kappa} \log \left( \sum_l e^{-\kappa z_l} \right) = -\psi^{-1} \left( \sum_l \psi(-z_l) \right)$$

where  $\min_l^{(\kappa)} \{z_l\} \leq \min_l \{z_l\}$  with equality attained in the limit as  $\kappa \rightarrow \infty$ .

We assume transmission over a  $q$ -ary input memoryless channel and also assume a corrupted codeword  $\mathbf{y} = (y_1, y_2, \dots, y_n) \in \Sigma^n$  has been received. Here, the channel output symbols are denoted by  $\Sigma$ . Based on this, we define a vector  $\boldsymbol{\lambda} = (\lambda^{(\alpha)})_{\alpha \in \mathbb{R}^-}$  where, for each  $y \in \Sigma$ ,  $\alpha \in \mathbb{R}^-$ ,

$$\lambda^{(\alpha)} = \log \left( \frac{p(y|0)}{p(y|\alpha)} \right).$$

Here  $p(y|c)$  denotes the channel output probability (density) conditioned on the channel input.

### III. LOW COMPLEXITY LP DECODING OF NONBINARY LINEAR CODES

To develop a low complexity LP solver for nonbinary linear codes, the present authors in [5] proposed a primal LP formulation which is equivalent to the original LP formulation proposed in [4]. This primal LP formulation has an advantage that, it has one-to-one corresponding with the Forney-style factor graph of the code and can be used to derive a suitable dual LP (see section IV in [5]). The dual LP is then “softened”

by using the “soft-min” operator which is used to derive the update equations given in *Lemma 6.1* in [5]. The softened dual LP is given below.

**SDNBLPD:**

$$\begin{aligned} \max. \quad & \sum_{i \in \mathcal{I}} \hat{\phi}_i + \sum_{j \in \mathcal{J}} \hat{\theta}_j \\ \text{Subject to} \quad & \hat{\phi}_i \leq \min_{\mathbf{a} \in \mathcal{A}_i}^{(\kappa)} \langle -\hat{\mathbf{u}}_i, \Xi(\mathbf{a}) \rangle \quad (i \in \mathcal{I}), \\ & \hat{\theta}_j \leq \min_{\mathbf{b} \in \mathcal{C}_j}^{(\kappa)} \langle -\hat{\mathbf{v}}_j, \Xi(\mathbf{b}) \rangle \quad (j \in \mathcal{J}), \\ & \hat{\mathbf{u}}_{i,j} = -\hat{\mathbf{v}}_{j,i} \quad ((i, j) \in \mathcal{E}), \\ & \hat{\mathbf{u}}_{i,0} = -\hat{\mathbf{f}}_i \quad (i \in \mathcal{I}), \\ & \hat{\mathbf{f}}_i = \boldsymbol{\lambda}_i \quad (i \in \mathcal{I}). \end{aligned}$$

The update equation can be used to update the dual variable  $\hat{u}_{i,j}^{(\alpha)}$  related to an edge  $(i, j) \in \mathcal{E}$  while all other edge variables are held constant. The updated value of the  $\hat{u}_{i,j}^{(\alpha)}$  is given by

$$\bar{u}_{i,j}^{(\alpha)} = \frac{1}{2} ((V_{i,\bar{\alpha}} - V_{i,\alpha}) - (C_{j,\bar{\alpha}} - C_{j,\alpha}))$$

where,

$$\begin{aligned} V_{i,\bar{\alpha}} &\triangleq -\min_{\substack{\mathbf{a} \in \mathcal{A}_i \\ a_j \neq \alpha}}^{(\kappa)} \langle -\hat{\mathbf{u}}_i, \Xi(\mathbf{a}) \rangle, \\ V_{i,\alpha} &\triangleq -\min_{\substack{\mathbf{a} \in \mathcal{A}_i \\ a_j = \alpha}}^{(\kappa)} \langle -\hat{\mathbf{u}}_i, \Xi(\mathbf{a}) \rangle, \\ C_{j,\bar{\alpha}} &\triangleq -\min_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i \neq \alpha}}^{(\kappa)} \langle -\hat{\mathbf{v}}_j, \Xi(\mathbf{b}) \rangle, \\ C_{j,\alpha} &\triangleq -\min_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i = \alpha}}^{(\kappa)} \langle -\hat{\mathbf{v}}_j, \Xi(\mathbf{b}) \rangle. \end{aligned}$$

Here the vector  $\tilde{\mathbf{u}}_i$  is the vectors  $\hat{\mathbf{u}}_i$  where the subvector  $\hat{\mathbf{u}}_{i,j}$  is excluded. Similarly vector  $\tilde{\mathbf{v}}_j$  is obtained by excluding the subvector  $\hat{\mathbf{v}}_{j,i}$  from  $\hat{\mathbf{v}}_j$ . Vector  $\tilde{\mathbf{a}}$  is same as  $\mathbf{a}$  where the  $j$ -th position is omitted and vector  $\tilde{\mathbf{b}}$  is obtained by excluding the  $i$ -th position from  $\mathbf{b}$ . Now by updating all the edges  $(i, j) \in \mathcal{E}$  with some schedule (e.g. circular), the low-complexity LP decoding algorithm converges to the maximum of the **SDNBLPD**. (see *Lemma 6.2* in [5]). The overall complexity of this algorithm is linear in the block length.

The terms  $(V_{i,\bar{\alpha}} - V_{i,\alpha})$  and  $(C_{j,\bar{\alpha}} - C_{j,\alpha})$  are related to the variable node (VN)  $i \in \mathcal{I}$  and check node (CN)  $j \in \mathcal{J}$  respectively. In the binary case, these terms can be efficiently calculated with the VN and CN calculations of the binary Sum-Product (SP) algorithm respectively [3]. However, for nonbinary codes, the calculation of  $(V_{i,\bar{\alpha}} - V_{i,\alpha})$  and  $(C_{j,\bar{\alpha}} - C_{j,\alpha})$  is not related to the VN and CN calculations of the nonbinary SP algorithm [5]. Hence, the CN calculations are carried out by processing exhaustively all of the possible codewords of the SPC code  $\mathcal{C}_j$ . Consequently, the complexity of calculating  $(C_{j,\bar{\alpha}} - C_{j,\alpha})$  (i.e of CN calculation) is in exponential in the maximum check-node degree  $d$ .

#### IV. MODIFIED BCJR ALGORITHM FOR CHECK NODE CALCULATION OF THE LOW COMPLEXITY LP DECODING

In [5], the authors suggested that the equations for  $C_{j,\bar{\alpha}}$  and  $C_{j,\alpha}$  can be rewritten as follows:

$$\psi(C_{j,\bar{\alpha}}) = \sum_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i \neq \alpha}} \psi(\langle \hat{\mathbf{v}}_j, \Xi(\mathbf{b}) \rangle) \quad (1)$$

$$\psi(C_{j,\alpha}) = \sum_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i = \alpha}} \psi(\langle \tilde{\mathbf{v}}_j, \Xi(\tilde{\mathbf{b}}) \rangle) \quad (2)$$

It may be observed from the above equations that the calculation of the  $C_{j,\bar{\alpha}}$  and  $C_{j,\alpha}$  is in the form of the marginalization of a product of functions. Hence it is possible to compute  $C_{j,\bar{\alpha}}$  and  $C_{j,\alpha}$  with the help of a trellis based variant of the SP algorithm (i.e. BCJR-type algorithm). One possibility is to use the trellis of the binary nonlinear code  $\mathcal{C}_j^{NL} = \{\Xi(\mathbf{b}) : \forall \mathbf{b} \in \mathcal{C}_j\}$ . However, due to nonlinear nature of this binary code, the state complexity at the center of its trellis would be exponential in  $d_j$ . Here state merging is also not possible. Hence there is no complexity advantage when we use the trellis of the binary nonlinear code  $\mathcal{C}_j^{NL}$ .

However if the trellis for the nonbinary SPC code  $\mathcal{C}_j$  is used, then the state complexity at each trellis step is  $\mathcal{O}(q)$  and is independent of  $d_j$ . The branch complexity of this trellis is  $\mathcal{O}(q^2)$ . In the following, we prove that the marginals  $C_{j,\bar{\alpha}}$  and  $C_{j,\alpha}$  can be efficiently calculated with some modifications to the BCJR algorithm which uses the trellis of the nonbinary code  $\mathcal{C}_j$ . For this purpose we define the following for the trellis of the SPC code  $\mathcal{C}_j$ :

- 1) The set of all states at time  $t$ ,  $\mathcal{S}_t, t \in (0, \dots, d_j)$
- 2)  $(s, s') \in (\mathcal{S}_t, \mathcal{S}_{t+1})$  represents a branch in the trellis which is related to the symbol  $b_t = s' - s$ .
- 3) Since we have trellis for SPC code, each state  $s \in \mathcal{S}_t$  represents the sum of all symbols from  $b_0$  to  $b_{t-1}$ .
- 4) We define

$$\sigma(i, j) = \sum_{r=i}^{r=j} b_r, \quad \mathbf{b} \in \mathcal{C}_j.$$

- 5) Branch metric for each  $(s, s') \in (\mathcal{S}_t, \mathcal{S}_{t+1})$  is  $g(s, s') = g(b_t) = \psi(\langle \hat{\mathbf{v}}_j, \xi(b_t) \rangle)$ .
- 6) State metric for forward recursion,

$$\mu_i(s) = \sum_{\substack{(b_0, \dots, b_{i-1}) \\ \sigma(0, i-1) = s}} \prod_{t=0}^{i-1} g(b_t), \quad s \in \mathcal{S}_i, i \in \mathcal{I}_j \quad (3)$$

$$\text{with } \mu_0(0) = 1, \quad \mu_0(\alpha) = 0, \forall \alpha \in \mathbb{R}^-.$$

and state metric for backward recursion,

$$\nu_i(s) = \sum_{\substack{(b_i, \dots, b_{d_j-1}) \\ \sigma(i, d_j-1) = s}} \prod_{t=i}^{d_j-1} g(b_t), \quad s \in \mathcal{S}_i, i \in \mathcal{I}_j \quad (4)$$

$$\text{with } \nu_{d_j}(0) = 1, \quad \nu_{d_j}(\alpha) = 0, \forall \alpha \in \mathbb{R}^-.$$

*Lemma 4.1:*  $C_{j,\alpha}$  and  $C_{j,\bar{\alpha}}$  can be efficiently computed on the trellis of the nonbinary code  $\mathcal{C}_j$  as follows,

$$\psi(C_{j,\bar{\alpha}}) = \sum_{\substack{(s, s') \in (\mathcal{S}_i, \mathcal{S}_{i+1}) \\ s' - s \neq \alpha}} \mu_i(s) \cdot \nu_{i+1}(s') \cdot g(s' - s) \quad (5)$$

$$\psi(C_{j,\alpha}) = \sum_{\substack{(s, s') \in (\mathcal{S}_i, \mathcal{S}_{i+1}) \\ s' - s = \alpha}} \mu_i(s) \cdot \nu_{i+1}(s') \quad (6)$$

where state metrics  $\mu_i$  and  $\nu_{i+1}$  are calculated recursively from previous state metrics via

$$\mu_i(s) = \sum_{b_{i-1} \in \mathbb{R}} \mu_{i-1}(s - b_{i-1}) \cdot g(b_{i-1}),$$

$$\nu_i(s) = \sum_{b_i \in \mathbb{R}} \nu_{i+1}(s - b_i) \cdot g(b_i).$$

*Proof:* First we prove that the state metrics can be computed recursively. The following may be observed from the definition of  $\mu_i(s)$ ,

$$\begin{aligned} \mu_i(s) &= \sum_{\substack{(b_0, \dots, b_{i-1}) \\ \sigma(0, i-1) = s}} \prod_{t=0}^{i-1} g(b_t) \\ &= \sum_{\substack{(b_0, \dots, b_{i-1}) \\ \sigma(0, i-2) + b_{i-1} = s}} \left( \prod_{t=0}^{i-2} g(b_t) \right) \cdot g(b_{i-1}) \\ &= \sum_{b_{i-1} \in \mathbb{R}} \left( \sum_{\substack{(b_0, \dots, b_{i-2}) \\ \sigma(0, i-2) = s - b_{i-1}}} \prod_{t=0}^{i-2} g(b_t) \right) \cdot g(b_{i-1}) \\ &= \sum_{b_{i-1} \in \mathbb{R}} \mu_{i-1}(s - b_{i-1}) \cdot g(b_{i-1}) \end{aligned}$$

Hence  $\mu_i(s)$  can be calculated recursively from the previous state metrics. Similarly, we can prove that the  $\nu_i(s)$  can be calculated from previous state metrics.

Now we prove the other part of the lemma. For ease of exposition we assume  $\mathcal{I}_j = \{0, \dots, d_j - 1\}$  in the following.

$$\begin{aligned} \psi(C_{j,\bar{\alpha}}) &= \sum_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i \neq \alpha}} \psi(\langle \hat{\mathbf{v}}_j, \Xi(\mathbf{b}) \rangle) \\ &= \sum_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i \neq \alpha}} \psi \left( \sum_{t=0}^{d_j-1} \langle \hat{\mathbf{v}}_j, \xi(b_t) \rangle \right) \\ &= \sum_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i \neq \alpha}} \left( \prod_{t=0}^{d_j-1} \psi(\langle \hat{\mathbf{v}}_j, \xi(b_t) \rangle) \right) \\ &\Rightarrow \psi(C_{j,\bar{\alpha}}) = \sum_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i \neq \alpha}} \left( \prod_{t=0}^{d_j-1} g(b_t) \right) \quad (7) \end{aligned}$$

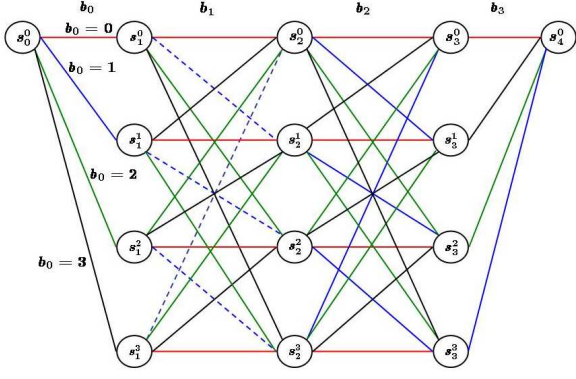


Fig. 1. States connected by dotted branches are used for the calculation of the  $C_{j,1}$ .

The right-hand side of (5) is,

$$\begin{aligned}
 & \sum_{\substack{(s,s') \in (\mathcal{S}_i, \mathcal{S}_{i+1}) \\ s' - s \neq \alpha}} \mu_i(s) \cdot \nu_{i+1}(s') \cdot g(s' - s) \\
 &= \sum_{\substack{(s,s') \in (\mathcal{S}_i, \mathcal{S}_{i+1}) \\ s' - s \neq \alpha}} \left( \sum_{\substack{(b_0, \dots, b_{i-1}) \\ \sigma(0, i-1) = s}} \prod_{t=0}^{i-1} g(b_t) \right) \\
 & \quad \left( \sum_{\substack{(b_{i+1}, \dots, b_{d_j-1}) \\ \sigma(i+1, d_j-1) = s'}} \prod_{t=i+1}^{d_j-1} g(b_t) \right) \cdot g(b_i) \\
 &= \sum_{\substack{(s,s') \in (\mathcal{S}_i, \mathcal{S}_{i+1}) \\ s' - s \neq \alpha}} \sum_{\substack{(b_0, \dots, b_{i-1}, \dots, b_{d_j-1}) \\ \sigma(0, i-1) = s, \sigma(i+1, d_j-1) = s'}} \\
 & \quad \left( \prod_{t=0}^{i-1} g(b_t) \cdot \prod_{t=i+1}^{d_j-1} g(b_t) \right) \cdot g(b_i) \\
 &\Rightarrow \sum_{\substack{(s,s') \in (\mathcal{S}_i, \mathcal{S}_{i+1}) \\ s' - s \neq \alpha}} \mu_i(s) \cdot \nu_{i+1}(s') \cdot g(s' - s) \\
 &= \sum_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i \neq \alpha}} \left( \prod_{t=0}^{d_j-1} g(b_t) \right) \quad (8)
 \end{aligned}$$

Using (8) in (7) we get (5). Equation (6) can be proved in a similar manner. ■

The overall algorithm works in two phases: in the first phase, the forward and backward state metrics are calculated and stored; in the second phase the marginals  $C_{j,\alpha}$  and  $C_{j,\bar{\alpha}}$  are computed with Lemma 4.1 where the state metrics computed in first phase are utilized. It may be observed that the aforementioned algorithm is essentially the same as the BCJR algorithm except for the second phase where marginals are calculated.

The calculations of the Lemma 4.1 can be visualized with the help of the trellis diagram. Figures 1 and 2 shows the trellis for the nonbinary SPC code of length 4 which is defined over

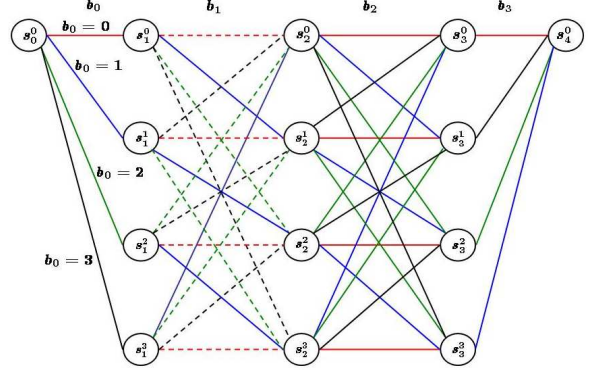


Fig. 2. States connected by dotted branches are used for the calculation of the  $C_{j,\bar{1}}$ .

$\mathbb{Z}_4$ .  $b_0$  to  $b_3$  represent the symbols, and states are represented by  $s_t^i$ , where  $t$  indicates the symbol after which the state occurs and  $i$  represents the sum of the symbols from  $b_0$  to  $b_{t-1}$ . The dotted branches in Figure 1 represents the transitions related to the symbol  $b_1 = 1$ . The state pairs which are connected by these branches are used for the calculation of the  $C_{j,1}$ . Similarly, the dotted branches in Figure 2 represent transitions related to the symbol  $b_1 \neq 1$ . Here the metrics of the corresponding state pairs are used for the calculation for the  $C_{j,\bar{1}}$ .

#### A. Alternative State Metric for Faster Calculation of $C_{j,\bar{\alpha}}$

The forward state metric  $\mu$  as defined in (3) needs to be computed for the calculation of  $C_{j,\alpha}$  and can be reused for the calculation of  $C_{j,\bar{\alpha}}$ . In (5) the algorithm needs to go through all branches  $(s, s') \in (\mathcal{S}_i, \mathcal{S}_{i+1})$ ,  $s' - s \neq \alpha$  for the calculation of  $C_{j,\bar{\alpha}}$ . If the proposed algorithm is implemented in hardware or on multicore architectures, then the computation time for  $C_{j,\bar{\alpha}}$  can be reduced by parallelizing its calculation. One possibility to parallelize calculation of  $C_{j,\bar{\alpha}}$  is to define a new forward state metric  $\bar{\mu}$ , which can be computed in parallel with  $\mu$  in the first phase and reduces the calculations required during the second phase of the algorithm. For this we define an alternative forward state metric as follows,

$$\bar{\mu}_i(s, \alpha) = \sum_{\substack{(b_0, \dots, b_{i-1}) \\ \sigma(0, i-1) = s, b_{i-1} \neq \alpha}} \prod_{t=0}^{i-1} g(b_t), \quad s \in \mathcal{S}_i, i \in \mathcal{I}_j, \alpha \in \mathbb{R}^- \quad (9)$$

with  $\bar{\mu}_0(s, \alpha) = 0, \forall s \in \mathcal{S}_0, \forall \alpha \in \mathbb{R}^-$ .

It should be noted that due to the condition  $b_{i-1} \neq \alpha$ ,  $\bar{\mu}_i(s, \alpha)$  cannot be calculated recursively from  $\bar{\mu}_{i-1}$ ; instead it is calculated together with  $\mu_i(s)$  from  $\mu_{i-1}$  as follows,

$$\bar{\mu}_i(s, \alpha) = \sum_{b_i \in \mathbb{R} \setminus \{\alpha\}} \mu_{i-1}(s - b_i) \cdot g(b_i)$$

With the help of the alternative forward state metric given in equation (9), the expression (5) of Lemma 4.1 can be rewritten as

$$\psi(C_{j,\bar{\alpha}}) = \sum_{s' \in \mathcal{S}_{i+1}} \bar{\mu}_{i+1}(s', \alpha) \cdot \nu_{i+1}(s') \quad (10)$$

The forward state metric  $\bar{\mu}_i(s, \alpha)$  requires the calculation and storage of an additional  $q - 1$  values for each state  $s \in \mathcal{S}_i$  during the first phase. Hence the storage requirement for the calculation of  $C_{j,\bar{\alpha}}$  with (10) increases by a factor of  $q$ . However, all additional state metric values can be calculated in parallel with  $\mu$  which does not effect the run time of the first phase of the algorithm. Also, the second phase of the algorithm needs to go through only  $q$  states instead of  $q(q - 1)$  branches, hence the overall run time for computing  $C_{j,\bar{\alpha}}$  is reduced with the state metric  $\bar{\mu}$ .

### B. Calculation of Marginals with $\kappa \rightarrow \infty$

In Lemma 4.1,  $\kappa$  is assumed to be finite. However, for many practical applications we are interested in  $\kappa \rightarrow \infty$ . According to Lemma 6.3 of [5], for  $\kappa \rightarrow \infty$  we again need to calculate  $(C_{j,\alpha} - C_{j,\bar{\alpha}})$  to update the corresponding variables. However, the marginals  $C_{j,\alpha}$  and  $C_{j,\bar{\alpha}}$  are here obtained as the limit of equation (2) and (1) respectively as  $\kappa \rightarrow \infty$ , i.e.,

$$C_{j,\alpha} \triangleq - \min_{b_i = \alpha} \langle -\tilde{v}_j, \Xi(\tilde{\mathbf{b}}) \rangle, \quad C_{j,\bar{\alpha}} \triangleq - \min_{\substack{\mathbf{b} \in \mathcal{C}_j \\ b_i \neq \alpha}} \langle -\tilde{v}_j, \Xi(\mathbf{b}) \rangle \quad (11)$$

Thus  $C_{j,\alpha}$  and  $C_{j,\bar{\alpha}}$  can be obtained by replacing all “product” operations with “sum” operations and similarly by replacing all “sum” operations with “min” operations in (2) and (1) (marginals with finite  $\kappa$ ). In (2) and (1) the marginalization is performed in the sum-product semiring. However for  $\kappa \rightarrow \infty$  the marginalization is performed in the min-sum semiring and hence the marginals of (11) can be computed with a trellis based variant of the min-sum algorithm. If we redefine the branch metric as  $g(b_t) = \langle \tilde{v}_{j,i}, \xi(b_t) \rangle$  and replace all “product” operations with “sum” operations and similarly replace all “sum” operations with “min” operations in equation (3), (4), (6), (9) and (10) then the resulting equations can be used on the trellis of the nonbinary SPC code  $\mathcal{C}_j$  to compute the marginals of (11). This trellis based variant of the min-sum algorithm is related to the Viterbi algorithm.

## V. RESULTS

This section presents simulation results for low complexity LP decoding which uses the trellis based check node calculations described above. We consider  $\kappa \rightarrow \infty$  for all simulations. We use the binary (504, 252) and (1008, 504) MacKay LDPC codes, but with parity-check matrix entries taken from  $\mathbb{Z}_4$  instead of  $GF(2)$ . These LDPC codes are (3, 6)-regular codes; hence there are 6 nonzero entries in each row of their parity-check matrix. We set the second and third nonzero entry in each row to 3, and all other nonzero entries are set to 1. Furthermore, we assume transmission over the AWGN channel where nonbinary symbols are directly mapped to quaternary phase-shift keying (QPSK) signals. We simulate up to 100 frame errors per simulation point.

The error-correcting performance of the (504, 252) and (1008, 504) LDPC code is shown in Figure 3 where the frame error rate (FER) of the LCLP decoding algorithm is compared with that of the min-sum (MS) algorithm. The MS algorithm also uses the trellis of the nonbinary SPC code for check node

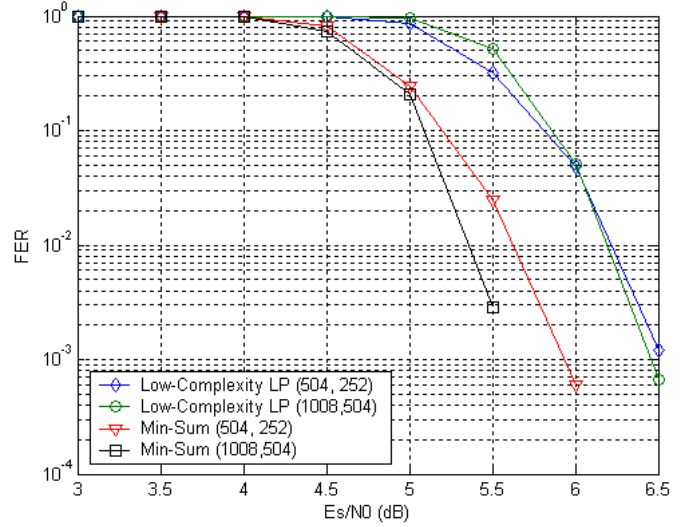


Fig. 3. Frame Error Rate for (504, 252) and (1008, 504) quaternary LDPC code under QPSK modulation. The performance of Low complexity LP decoding is compared with that of the min-sum algorithm.

processing. The maximum number of iterations is set to 64 for both decoding algorithms. For the (504, 252) code, the FER of low complexity LP decoding is within 0.5 dB from that of MS algorithm and for (1008, 504) code, it is within 0.7 dB. These results are comparable to that of the binary LCLP decoding algorithm of [3]. Finally, it is important to note that these LDPC codes are significantly longer than the quaternary (80, 48) LDPC code tested in [5].

## VI. CONCLUSION

In this paper, we proposed a modified BCJR algorithm for efficient check node processing in the nonbinary LCLP decoding algorithm. The proposed algorithm has complexity linear in the check node degree. We also proposed an alternative state metric which can be used to reduce the run time of the proposed algorithm.

## VII. ACKNOWLEDGMENTS

The authors would like to thank P. O. Vontobel for many helpful suggestions and comments. This work was supported in part by the Claude Shannon Institute, UCD, Ireland.

## REFERENCES

- [1] M. C. Davey and D. J. C. MacKay, “Low density parity check codes over  $GF(q)$ ,” *IEEE Communication Letters*, vol. 2, no. 6, pp. 165–167, June 1998.
- [2] J. Feldman, M. J. Wainwright and D. R. Karger, “Using linear programming to decode binary linear codes,” *IEEE Transactions on Information Theory*, vol. 51, no. 3, pp. 954–972, March 2005.
- [3] P. O. Vontobel and R. Koetter, “Towards low-complexity linear-programming decoding,” in *Proc. of 4th International Conference on Turbo Codes and Related Topics*, Munich, Germany, April 3–7, 2006.
- [4] M. F. Flanagan, V. Skachek, E. Byrne, and M. Greferath, “Linear-Programming Decoding of Nonbinary Linear Codes,” *IEEE Transactions on Information Theory*, vol. 55, no. 9, pp. 4134–4154, September 2009.
- [5] M. Punekar and M. F. Flanagan, “Low Complexity LP Decoding of Nonbinary Linear Codes,” *The Forty-Eighth Annual Allerton Conference on Communication, Control, and Computing*, September 29 – October 1, 2010.